IRM Software Features

Robert Goodwin

September 18, 1997

Internet Rack Monitor software is a large topic that has been described via numerous detailed documents available on the Web. This document is meant to help introduce the IRM software to those interested in learning how the system operates.

The features included in IRM software can be divided into the following categories:

- General
- System tables
- Clock events
- Data pool processing
- Device control support
- Local application support
- Alarm handling
- Page application support
- Network support
- Diagnostics

There are many relationships between these categories. System tables support many of the other feature categories. Network protocol support accesses the data pool and can effect device control. Alarm handling compares data pool values with reference values in system tables. Page and Local applications reference and modify the data pool. Diagnostics assists in debugging and monitoring the health and configuration of the entire system.

General—Characteristics

IRM software is designed as a front end that operates in real time, performing all its assigned tasks at 10–15 Hz, according to a synchronizing external clock interrupt. A number of IRMs, connected via the network, comprise a project, all nodes synchronized to perform their tasks together. Each node is thus one part of a distributed control system. All nodes run identical system software, so that all features are supported by all nodes.

General—Hardware

Each IRM includes a MVME-162 CPU board with 25 MHz M68040 CPU, 4 MB RAM, 1 MB ROM, 1 MB flash memory, 0.5 MB non-volatile RAM, two serial ports, ethernet controller, VMEbus interface, and four sockets for IndustryPack boards. One IP socket contains a digital board, required in each IRM, that connects via an off-board digital interface to 8 bytes of digital I/O (via four connectors) and also decodes a "Tevatron clock" signal to derive the synchronizing 10–15 Hz interrupt that drives the system's cyclic operation. This same board can also support a second 8-byte digital interface card, if needed. Another IP socket connects via an off-board analog interface to 64 channels of A/D (via four connectors) and 8 channels of D/A (via two connectors). Two IP sockets are then available for other boards, such as an 8-channel timer or an 8-channel Swift digitizer (that provides 8 channels of clock event-triggered waveform data captured at rates of 6–800 KHz), or even another 64-channel A/D analog board.

Connectors at the rear of the IRM are for the 64 A/D signals, the 8 bytes of digital I/O, the 8 D/A channels, the Tevatron clock, and the ethernet network. When additional IP boards are used, additional rear panels provide the appropriate connectors.

At the front of the IRM is a diagnostic software test points connector, 8 LEDs, 8 option switches, two serial port connectors, and the CPU board reset and abort switches. The test points connector, suitable for use with a logic state analyzer, includes signals that monitor the timing activity of 14 tasks and 10 interrupts. Eight of the interrupt signals also appear visually on the LEDs.

The CPU board occupies one slot of a 3-slot VMEbus crate inside the IRM chassis. The CPU board alone has been enough to support the IRMs installed so far. In fact, it was the greatly increased functionality provided by the 162 CPU board that enabled the IRM single-card concept to take shape. (Previous VME incarnations required 4–5 boards per node.)

General—Synchronization of tasks

An interrupt from the digital IP board, set to occur at a delay from a selected clock event, is used to wake up the Update task to begin each cycle's activity. Update first interprets the list of binary commands in the Data Access Table to update the data pool. Many possible commands can access and modify data from various hardware interface types. One command copies out the latest set of 64 A/D readings from the IP board memory into the data pool. (The IP board digitizes all 64 channels at 1000 Hz, so there is no waiting for the digitization process to take place; the analog signals have already been digitized and stored even before the cycle interrupt occurred.) The 8 bytes of digital data are read from the digital IP board. Modify commands adjust the data pool values as needed for the devices connected to that particular node. Another command causes each enabled local application to be invoked to perform whatever functions are needed for the present cycle. Each local application maintains its own context for as long as it is active, which is usually as long as the IRM is powered on and the system is running. In this way, local applications are a kind of catch-all solution that allows support of anything not covered by the usual built-in Data Access Table commands.

After the data pool has been updated for the present cycle, the Update task builds reply messages for all non-server-mode data requests that are due on that cycle and queues them to the network. The linked list of active requests is maintained in order of requesting node so that multiple replies destined for the same node can be combined into common frames to increase network utilization efficiency.

When the Update task finishes its work, the Alarms task scans all analog channels and all digital bits in the data pool for changes of alarm state (good/bad) and builds any change-of-state messages for prompt delivery to an alarm target node#, usually an IP multicast network address.

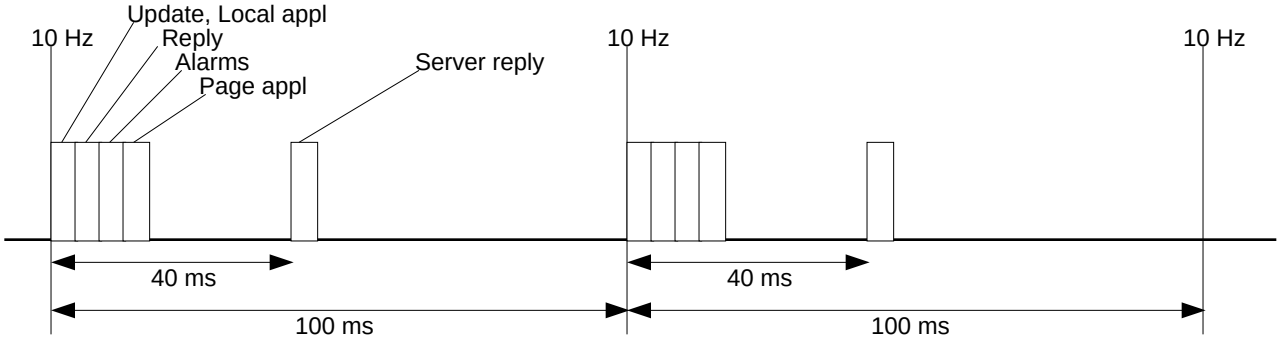
Upon completion of the alarm scan, the Page Application task invokes the active page application. It may update its virtual screen, for example, which may be viewed by other nodes via the "Page G" connection, described later. A page application maintains a a simple 16-line by 32-character virtual user interface in lieu of a command line interface.

Besides the three tasks above, additional activities that may occur during the cycle include various kinds of network communication. Network messages may be received at any time. A one-shot data request results in an immediate reply without waiting for the next cycle.

At 40 ms beyond the start of the current cycle, any server-type replies are updated and queued to the network. A server-type request is one that requests data to be retrieved from one or more *other* nodes. Each node contributing to such a request will, of course, deliver its replies to the server node early in the cycle as soon as its own data pool has been updated; thus, 40 ms represents a kind of deadline by which time such contributing replies must be received by a server node so that they can be put together and delivered to the original requesting node. In this way, the composite reply includes correlated data that was measured at the same time across a number of nodes in the net-

work. It is for the purpose of being able to provide correlated data in replies that all nodes are operated by a synchronous timing signal.

The following schematic illustrates the sequence of activities that occur each cycle.



In this diagram, the relative times shown are *not* to scale. Typically, the times for performing the activities shown are much shorter than indicated here. Viewing what the real times are can be done with a scope that connects to the task activity signals on the test points connector on the IRM front panel.

System tables—Introduction

Most system tables are placed in non-volatile memory, so that their contents survive resets and power outages. Many tables included device-related data, either for analog channels or digital bytes. The key to an analog channel is a channel#, usually in the hexadecimal range `0000–03FF`. The key to digital data may be a byte# or a bit#. Byte#s usually range from `0000–007F`, and bit#s correspondingly range from `0000–03FF`. A key is a index into system tables that are organized as arrays of records. A system table directory occupies the first part of the non-volatile memory, wherein each table is characterized by its base address, its number of records, and the size of each record.

System tables—Analog channels

Characteristics of an analog channel, as known via its channel# key, depend upon values held by the relevant system tables in the channel#-indexed records. For example, a channel# key references a 6-character device name, 4-character scaled units text, linear scaling constants for reading and setting, and specifications for analog control and related digital status and control. Having this information in a local database allows support of meaningful page applications, encoded alarm messages, and local applications that perform their logic in scaled units. It also permits a given control system implementation to operate without recourse to any particular commercial database. Details of the form of control associated with an analog channel need only be stored locally.

An analog channel may be both readable and settable, even though different hardware is involved in providing data readings versus control settings. For example, a magnet power supply controlled by a D/A produces current that passes through a magnet. The current is measured by a transducer. The channel's reading gives the present value of the current; the channel's setting, in the same scaled units, gives the expected value of the current according to the accuracy of the scaled constants used for *setting*. If the reading of the channel is 52.3 amps, but it is desired to set it to 60 amps, the scaled constants for *setting* are used to derive a D/A value to be sent to the power supply; the scaled constants for *reading* interpret the transducer's digitized value to derive a value of the current. If this result is 61.1 amps, then the scaled constants for *setting* may be slightly inaccurate. The reading is 61.1 amps, to the accuracy of the scaled constants for *reading*. If 60 amps is needed, the D/A setting should be adjusted a little until the reading shows 60 amps. The low level software does not iterate

on such control actions. If such iteration is desirable in particular cases, it can always be handled via a local application, in which the setting would control a dummy value that the local application monitors, adjusting the D/A output voltage as necessary until the A/D reading matches within some desired error band. Often, control actions for such devices are made by manual tuning, so the whole matter isn't an issue. One observes the reading of the power supply as one adjusts the D/A incrementally, stopping when the reading appears satisfactory—or when some other condition is reached, such as maximum accelerator beam intensity downstream of the magnet. It is the *reading* value that is meaningful. The *setting* value is only that value which produces the reading value, even when the scaled constants for setting produce a value different from the reading value.

Populating analog device-related tables can be done easily via the Analog Page application. In a large control system that uses a central database, it may be desirable to download central database changes to insure that the two databases track. It is important that they *do* track, because the detailed means of controlling an analog channel most immediately depends upon the values in the *front end* database, not those stored in a central database record. When a host computer issues a setting to an analog channel, it specifies the channel# key and the data to be sent; it does not specify the details of how the setting will be performed, what hardware will be addressed, what multiplex selection, etc.

Commands placed as entries in the Data Access Table determine how an analog channel obtains an updated reading. A single command may affect a range of channels, such as that which copies 64 readings from the IRM A/D hardware memory into the data pool, which is indexed by the channel# key. An analog channel's setting value, the last value sent to the hardware, is also maintained. This allows support of incremental (knob) control that can permit multiple user access, in that each user's incremental actions accumulate in a rational way to produce the result received by the hardware. Of course, multiple user control may not be desired by administrative *fiat*, but at least this scheme permits it to work sensibly. Another use for the setting values is to allow a host to save machine conditions for later recall. The other reason for keeping the last setting values occurs at system reset time, when the setting values for all controllable analog channels are refreshed to the hardware D/A's. This is done because a reset may occur following a power outage, when D/A hardware register contents are lost.

Each analog channel may be enabled for alarm scanning, so that a nominal (reference) value and tolerance (dead band) may be defined. The Alarms task performs the scanning of each channel every cycle, emitting an alarm message for each alarm state change. A count of alarm state transitions is also maintained for diagnostic use. For a node that is configured for 1024 channels and 1024 bits, but with none enabled for alarm scanning, the scan takes about 1.1 ms. With 150 channels enabled, the scan time is about 1.4 ms.

System tables—Digital bytes

Digital status/control is organized into bytes, so that the key is a byte#. Each byte has an associated table entry that includes its reading value. One Data Access Table command refreshes all digital byte readings in the data pool. Each byte includes 8 bits, each of which can be referenced via a bit# key, so that 1024 bit#s map into 128 byte#s. Each bit# key references a 16-character title, and each bit# can be enabled for alarm scanning by the Alarms task. It is also possible to build composite status words that hold up to 16 bits, and treat that word as a digital status word to be included in the analog alarm scan. In that case, such "analog" channels are scanned using a nominal bit pattern and mask logic.

Digital status/control references can also be related to an analog channel. This allows, for example, a magnet power supply channel to exhibit on/off status, interlock status, and allow on/off and reset control actions. The key can be a channel#, which, via the analog system tables, refers to specific

bit#s. The type of digital control needed is also indicated there, whether it is a d.c. level, a pulse, or a pair of pulses.

System tables—Pages

Each display page is supported by a page application as specified in the page tables. Each page has an associated file name and a 16-character title. It also has a dedicated 128-byte table entry that is used to save parameters that are needed upon display call-up. The Memory Dump Page application, for example, must retain 8 node#s and addresses so that the next time it is invoked, the display reflects the saved parameter values. Another field supports the automatic invocation of a page application based upon periodic calendar time. For example, a page application might be called once a day, say, to collect some statistics and print out a formatted summary via the serial port.

System tables—Files

Another set of system tables supports a non-volatile memory-resident read-only file system. (Here, read-only means that one must rewrite an entire file to make even a small change.) Page applications and local applications are the most common files stored here. Each program is compiled and downloaded separately from the system code. When such a program is first initialized, its checksum is checked to insure that the non-volatile copy has not been corrupted, and the contents of the file are copied into allocated dynamic memory for execution. As a result, new versions of such code files can be downloaded even when the program is active. For local applications, when downloading one of the same name as one currently in execution, an orderly switch to the new version takes place as soon as downloading is complete. (For page applications, one must recall the page to make the switch.) Because local application enable bit values are kept in non-volatile memory, the same set of local applications are enabled following a reset of the node as the set that was enabled when the node was last running. In this way, local applications operate as an extension of the system code, even though they are separately maintained.

Local applications to be run via a special command in the Data Access Table, usually near the end, are registered in the LATBL system table. Each entry retains a set of 10 parameters that are passed to the local application when it is invoked, where the first one is always the enabling bit#. Setting the enable bit# activates the local application to be called every cycle. The first call is an initialization call that allows the application to allocate memory to maintain its context across calls. A pointer to this memory is also passed to the application each time it is called. When the enable bit# is turned off, a final call is made to the application, allowing it to release its context memory and generally "clean up its act." For example, one might have an LA that performs a closed loop function. If more than one device needs the same closed loop support, multiple instances of the LA are set up; each shares the same copy of the allocated executing code, but each has its own set of parameter and its own context memory.

System tables—Network

Several system tables help support network activities. One network table houses the node's physical network address and the multicast addresses available to it. It contains a network connection table that associates internal message queues and protocol handler tasks with UDP port#s or Acnet network task names. It also supports a token ring network interface if appropriate, although IRMs generally do not need token ring because an ethernet interface is included on the CPU board.

In order to provide Internet Protocol support, an ARP table is needed to hold physical addresses for active Internet nodes, plus a list of active UDP port#s for each such node. Via this table, a UDP socket can be encoded into a 16-bit "pseudo node#." There is a table that keeps a list of IP networks/hosts that are allowed setting access. Whenever a setting message is targeted to the node, it is denied if the source IP address does not find a match in this table. The TFTP server also checks this table be-

fore accepting a write file command.

Another table caches replies from the Domain Name Server that are needed to target nodes identified by a 16-bit node#. A local application called `DNSQ` maintains this table. Each IRM node is registered with the local DNS as `nodexxxx`, where `xxxx` is the hexadecimal value of the node#. For example, the IRM known at Fermilab as node# 509 has the Internet name `node0509.fnal.gov`. Another network table is used to hold IP addresses of Acnet nodes, since they do not use this scheme.

System tables—Data streams

The IRM system provides formalized support for circular buffers known as data streams. One standard system data stream is used for network frame diagnostics, where information about each network frame that is received or transmitted is recorded, including the node#, the frame size, a pointer to the frame buffer, and the calendar time that the frame was received or transmitted with a resolution of milliseconds from the start of the current cycle. Another system data stream is used to record setting commands as a diagnostic aid. Data stream queues are established in volatile memory during system initialization according to the parameters found in the non-volatile data stream table.

Clock events

The Tevatron clock system was designed for use at Fermilab to carry up to 256 synchronizing clock signals to all parts of the accelerator. Each 8-bit event# is encoded onto a 10 MHz serial carrier signal. This clock system technology is quite portable and has been found to be useful at other labs.

Hardware support for clock event decoding is included in the digital IP board that is a required part of an IRM configuration. An interrupt is generated for each clock event, and a 32-bit microsecond counter—part of the 162 board hardware—is read and recorded for that clock event, along with the elapsed time since the last such event. A 256-bit array is maintained that shows all the events that have occurred since the last cycle. If desired, interrupts may be masked from particular known high-rate events that are of no local interest. By sampling the clock event bit array, clock events can be used to influence Data Access Table command processing as well as the logic in various local applications. Classic protocol can return replies based upon whether a selected event occurred since the last cycle. The time to process each clock event interrupt is about 10 microseconds as measured at the software test points connector on the front panel of the IRM.

The Clock Events Page application shows the clock event activity experienced by any IRM node. It shows the average events/second rate, a histogram of the occurrence of single and multiple events, the activity of all clock events on a cycle-by-cycle basis, and for a selected event, its count, delta timing, and timing of its occurrence relative to the node's own cycle interrupt.

Data pool processing

Much of a system's device configuration is defined via the contents of the Data Access Table, introduced in the *General—Synchronization of tasks* section. The entries of this table are the commands that are interpreted early in each cycle by the Update task. A sequence of commands may be conditionally executed each cycle, every "n" cycles, or depending upon the reading of a specified bit# or analog channel#. Planning the contents of the DAT is done with some care. The end result is a list of commands that refresh the data pool. The Update task performs the interpretation of this table, during which time no other tasks can run, so that the entire data pool appears to update instantaneously; it is not possible for a network data request message to be received that retrieves values from the data pool at a time when it is only partially updated for a given cycle. This is done to ensure support of correlated data across multiple channels and bits and across multiple nodes.

Examples of DAT commands are as follows:

Copy from IRM A/D memory
Refresh digital data pool from byte address table
Copy from memory words/bytes
Perform addition, subtraction, multiply, divide of analog channel data
Pedestal subtraction depending on beam status
Derive cycle count since event/bit
RF detector diode linearization
De-multiplex data words/bytes
Copy memory into data stream
Format status words from selected bytes/bits
Capture all readings in present cycle
Invoke all enabled local applications

The last one listed is the catch-all command that provides for many specialized data pool updating needs. As an example, a PLC controller was purchased to interface to vacuum hardware. It supports a serial RS-232 connection using a specified protocol. A local application was written that is called not only every cycle, allowing for timeout logic, but also for every line of text received via the serial port. (There was too much data to access all of it every cycle, but that limitation was accepted once the serial-interfaced controller was selected.) It updates all such data every 2–3 cycles, and it sends out a setting command, if necessary, between updates. After this data becomes part of the data pool, it takes on all the built-in support granted analog channels and digital bytes.

No matter the source of the data in the data pool, it all looks the same once the data pool is updated. This greatly simplifies alarm scanning and updating of data replies. The commands for updating the data pool are designed to be executed efficiently, since this work is done every cycle, 24 hours/day. For example, one command updates all 64 consecutive analog channels from the A/D circular buffer memory on the analog IP board. If, instead, a database entry for each analog channel specified a data source field from which the value is to be copied, it would take much longer to update the 64 channel reading values. Also, it may be more difficult to ensure correlation between channels, as the A/D is always active. Array processing greatly increases efficiency. The IRMs used in the Fermilab Booster HLRF system finish execution of all DAT commands, including all enabled local applications, by 3 milliseconds into each cycle.

Device control support

This topic was earlier addressed in the section *System tables—Analog channels*. Many kinds of analog control device types are supported, each specified via an analog control type#. Those typically used with IRMs are as follows:

```
00: No control
02: Motor
05: Memory word (8-bit writes)
0A: Memory word (16-bit write)
0C: Unsigned 12-bit D/A
10. Memory byte (single byte no shift)
11. Memory byte (single byte w/ shift count)
12. Same channel reading word w/mask
15. IRM D/A with offset and 4-bit right shift
16. IRM D/A with offset but w/o right shift
17. Unipolar 15-bit D/A clamped at zero.
18. Unsigned 16-bit D/A or timer
19. PLC memory word via PLCQ message queue.
```

Most drivers are very simple accesses to memory with various formatting of the data word before it's actually written to a hardware register. The last one is more involved; it relates to an earlier example given for a PLC interface under the *Data pool processing* section, in which the interface is driven via an RS-232 serial port. To effect such D/A control, the command is passed through a message queue that is created by the `DNET` local application. When the local application has just completed a data acquisition, it checks the message queue; if a setting command is found there, it arranges to do the required serial transaction using the DirectNET protocol supported by the PLC controller. The serial transaction is slow, but overall system operation is not delayed by it as the serial transaction is interrupt driven. The interrupt routine takes about 3% of the CPU time at 19.2K baud.

Local application support

Local applications provide extensibility of the IRM system as mentioned above under *System tables—Files*. The code for a local application is prepared as a Pascal procedure, or as a C function that returns void. The library uses a Pascal calling sequence convention, so it is convenient to use a C compiler that includes the *pascal* keyword to select this option. Such is provided by the C compiler available with either Apple's Macintosh Programmer's Workshop (MPW) or MetroWork's Code Warrior development system. Each time the system invokes the local application, it passes two parameters. The first is a call type number and the second is a pointer to a structure that includes the pointer to the static variables allocated during the LA's initialization call to retain its context while it is enabled, followed by the 10 words of argument values that allow particularization of the LA's actions. The first of the 10 arguments is always the enabling bit# that causes the LA to be active. One Data Access Table command causes each LA *whose enable bit is set* to be invoked in turn. It also notices when an enable bit changes state. If it notices the bit just became set, it finds the code for the LA in the memory file system, evaluates its checksum to be sure the code has not been corrupted, allocates space for it in volatile memory, copies the code into this space, and invokes the code for the first time, specifying an initialization call type. The LA allocates a block of memory to maintain its executing context and deposits the pointer to this block into the structure passed to it. In every subsequent call, the LA receives this pointer so it can remember what it was doing. Subsequent calls occur every cycle as long as the enable bit remains set. If the enable bit is reset, then a final termination call is made, during which the LA cleans up its act, releasing the memory that it allocated during the initialization call.

Besides the initialization, termination and cycle calls, an LA may also be called in response to receipt of a network message for which it has set itself up to be the handler. A simple example of such would be the UDP Echo Server LA. It initializes itself to receive all UDP datagrams addressed to port 7. When such a datagram is received, this LA is called with the network call type. The LA then echoes to the sender's UDP port# the same datagram. In this simple example, the cycle calls are ignored by the LA.

Another simple example of an LA would be one that adds the readings of two analog channels to produce a result that is assigned to the reading of a third channel. (As this function is already included in the built-in data access table type, it would be better to use that. This is only an example.) The LA would use three argument words, in addition to the enable bit#, that would be set to the three channel#s. Each time the LA receives a cycle call, it would get the reading of the first two channel# arguments, install the sum as the reading of the third channel# argument and return.

Other examples of LAs are those used for closed loop applications, which can be simple or complex. One specific LA that is fairly complex supports the elaborate Acnet protocol used for Fast Time Plots and Snapshots, that known at Fermilab as FTPMAN.

Another example is a Network Time Protocol client LA. Every so often, according to one argument

value, it sends a network time request message to a destination network time protocol server node, specified by an IP address (that occupies two argument words). When the reply is received, the system again invokes this LA to process it. The LA interprets the reply message and establishes the time-of-day for the local system, optionally sharing such timely news by sending it to a target multicast node#, as specified by yet another argument word. In this way, the IRMs within one project can be kept in synchronization with respect to calendar time-of-day. Every few minutes, one IRM that runs this LA can collect the time of day and share it with the others. In between updates, each IRM is perfectly capable of counting its own time using its own crystal. This synchronization just protects against crystal differences and time lost due to power outage or when an IRM is reset.

When developing an LA, it is possible that a bug can cause an unexpected exception error to occur that causes the system to abort. If an abort occurs while executing the LA code, the enable bit for that LA is reset. As the IRM automatically reboots, the LA will be disabled, so that the abort will not recur.

A special page application called the Local Application Page provides convenient access to set up the arguments of a local application that are contained in entries of the LATBL. (In a sense, it is the user interface for all local applications that by definition have no user interface.) Each argument is shown with a keyword prompt and the name and value of any argument that is designated a channel or bit. Any such argument channel or bit value may also be set via this page application. It also monitors the present execution of the LA, showing the LA's version date and the address of the block of memory housing the execution context of the LA. Page applications are discussed later in this document.

In summary, LAs serve to add functionality and particularize a given IRM according to its unique needs. Each LA is separately compiled and downloaded to a specific IRM. The first time a version is downloaded via a TFTP client, it is stored in non-volatile memory and the current time-of-day is used as a version date. When it is subsequently copied to another node, using the Classic protocol, the version date is retained. In this way, one can see whether a given IRM is running the latest version of an LA. If not, it can easily be updated by copying from a node that has the latest version.

Alarm handling

Alarm scanning is performed every cycle by the Alarms task. All analog channels and all binary bits that have been enabled for alarm checking are checked after the data pool has been updated and all active requests have been answered. Times for an IRM to do this job are typically 1–2 ms, with additional time required if alarm messages need to be formatted for delivery to the network. The usual method is to target Classic protocol alarm messages to a multicast node#, so that any node reachable in that way can "listen" to these alarm messages.

Analog channels can be scanned for having a reading value within a specified tolerance range of a nominal value. As long as the reading remains inside the tolerance window, the channel is in the "good" state. When its reading wanders outside the range, it enters the "bad" state. Once in the "bad" state, it must come within half of the tolerance range before it can be again considered "good." This hysteresis logic prevents alarm message chattering when a reading hovers near the tolerance window edge. Some analog channels can be specified as containing digital status information. Such channel readings are derived from collecting certain bit#s together to form a digital word whose alarm state is to be checked as a whole. In this case, the nominal value field is interpreted as a nominal bit pattern, and the tolerance word is interpreted as a mask of the bits that are to be checked. If all bits in the reading word match the nominal bit pattern *within the mask*, then the channel is "good," else it is "bad." In this way, one alarm message can be generated for all such bits indicated in the mask. It is a kind of digital alarm scan, but for up to 16 bits taken together.

Binary bits can also be scanned as individual "devices." Each bit has an associated nominal state. If the present reading of the bit matches the nominal state, then the bit is "good," else it is "bad."

Whether an analog channel is checked, or a binary bit is checked, several options apply, according to the alarm flag bits associated with that channel or bit. Setting the "inhibit" bit will force a dedicated digital control line to be asserted as long as any so-marked channel or bit is in the "bad" state; this control line is normally used to inhibit beam. One may set a "beam" bit that enables alarm scanning for a channel or bit only on beam pulses. One may set a "silent" bit that inhibits sending an alarm *message* when the alarm state changes; this option is used to merely keep a diagnostic count of alarm state changes without bothering the operators. One may require that an out-of-tolerance condition exist for 1–16 consecutive cycles before declaring a device in a "bad" state, in order to filter noisy signals. While an up-to-16-cycle delay in *reporting* an alarm condition may not cause a problem for human operators, there will also be such a delay before beam can be inhibited, in the case that the beam inhibit option is selected. One should therefore use this option with care.

One other type of alarm message is supported, that of "comment" alarms. There are only two of these available so far. One occurs following reset of an IRM. The other occurs when an IRM performs an "alarm reset," in which all good/bad bits are forced to "good," and devices that were "bad" at the time of the reset cause "good" messages to be emitted. An alarm reset action may be needed when a project-wide alarm manager is restarted. Of course, an alarm reset action is performed internally when an IRM resets.

It is also possible to inhibit the reporting of *all* alarm messages—except the bit that selects this option, so that one has a way of being reminded that an IRM is in this state. This option can be used as a defensive tactic. With alarm scanning done every cycle, an IRM has the potential of emitting an enormous number of alarm messages that could overwhelm an alarm handler, depending on its design.

Besides reporting alarm messages to the network, ascii-encoded alarm messages can optionally be printed out the local serial port. If it is desired to collect such alarm messages from all nodes in a project, one can arrange to have one IRM enabled to receive the alarm messages that are multicast from all nodes in the project, then enable its serial output option. That IRM will then encode and print all alarm messages from all nodes in the project. Any that are generated by its own alarm scan, plus any that it receives from listening to the alarm multicast address, will be encoded for serial output. Storing such a serial stream in a host text file produces a simple alarm log for an entire project.

Special support is provided for Acnet alarm reporting. A local application AERS is used to shepherd alarm messages in the Acnet format to the alarm handling process on a central Vax. IRMs that don't use Acnet simply omit this LA.

Page application support

Page applications are similar to local applications in that they are separately compiled and downloaded into the non-volatile memory file system. But a page application provides a user interface, and only one page application can be active at a time. Through a page application's user interface, one may switch to another PA via an index page that lists all PAs that are available. The index page also provides for assigning a page to a specified PA. There are 31 possible pages available for such assignment, numbered 1–9, A–V. The execution context for a page application is established upon initialization of the PA when it is first invoked, just as for LAs. But in place of arguments in a LATBL entry, there are 120 bytes of page-private data that allow the PA to preserve its context across separate call-ups of that page. Establishing values for this context is done via the PA's own user interface. Often, a PA uses its page-private memory to retain enough context so that the page

display appears the same when it is next invoked as it appeared when it was last active.

As for LAs, a PA receives an initialization call when it is first invoked from the index page. Each call specifies two parameters: the call type and a pointer to the page-private memory for that page. Thereafter, it receives calls every cycle after alarm scanning has completed. When the user invokes a different page, or returns to the index page, a termination call is made so the PA can clean up its act. In addition, a PA can receive a "keyboard interrupt" call that has the meaning of initiating some action that is usually implied by the location of the cursor on the 16-line by 32-character page display.

Page applications were originally developed for use with a special hardware display and keyboard interface. But IRMs are not normally configured with this "little console" hardware that also requires a special Crate Utility VME board. Without this board, an IRM still supports the execution of page applications, but the 512-character display is maintained only in memory. A host computer that implements a "Page G" utility program can access this page memory and present it on a user's console. (The "Page G" label comes from its traditional assignment to that page on a "little console.") It monitors keyboard activity and sends appropriate settings to the IRM to make it think a character has been typed on its "little console," even though none exists. The host program emulates the "little console" in order to gain support of the functionality offered by the suite of available PAs. If two users, each with an emulator program, view the page display of the same IRM, then each sees the same display, and each can move the cursor and type on the display or initiate actions based upon same. This is sometimes a useful means of one user illustrating how to perform an action supported by a certain PA for a user in another office. Such "Page G" emulator programs have been written for the Macintosh, PC, Sun workstation, Vax, and IRM.

The software that runs in IRMs was originally developed in 1980. Only four PAs were available at that time, and all four still provide useful functionality. One is the Parameter Page that shows a user-specified list of up to 14 analog channel readings and allows control of same. Each channel's reading value is updated about once a second with the average of the readings measured over the last 13 cycles, preferentially using beam cycle readings, if any. Setting values and alarm nominal and tolerance values can be similarly viewed and controlled. To establish a parameter line, the user enters either a node# and channel#, or an analog device 6-character name. The name may either be from the local node, or from some other node in the project; in any case, the PA will learn its node# and channel#. To support a device name lookup data request for non-local names, the system multicasts a name lookup request, causing other IRMs in the project to search their own tables for a match; the IRM finding a match replies to the request. Name table search performance is enhanced by use of a hashing scheme.

The second original PA is the Analog Page. It allows configuration of a given node's analog channel database fields, including its name, descriptive text, units text, scale factors, analog control information, associated status/control bits, and alarm control flags. It can also print out the selected node's analog database or data pool via the local serial port.

The third original PA is the Binary Page. It allows configuration of the descriptive text and alarm control flags for each binary bit in a given node. It also allows for digital control, either by the bit or by the byte. It can also print out this information via the serial port.

The fourth original PA is the Memory Dump Page that permits viewing 64 bytes of memory, eight bytes per line, either contiguous or from 8 independent addresses. The display is updated every cycle, so that one has a live window view of memory that has proven itself invaluable for diagnostics. One can also change the memory that is viewed, by the byte or the word, merely by typing over the value displayed and initiating the "keyboard interrupt" action. If the memory setting is successful, the new

value is immediately refreshed.

From the beginning, page applications have been allowed access to devices or memory of non-local nodes via transparent use of the network. The Parameter Page can display local channel information, but it can just as easily display such information from any set of non-local nodes. The PA does this without consciously using the network, although that surely is what goes on in the background. The PA simply makes a data request for the devices that it wants; for each device, it specifies both the node and channel#, bit#, memory address. or other key. Later, it calls for the data that it had requested, and the system furnishes it, waiting, if necessary, for it to arrive via the network. The system code does whatever is required to obtain the data and supply it to the PA, which doesn't need to know or care whence it came.

Since the early days, many more PAs have been written to perform various system configuration and diagnostic functions. The Local Application Page was mentioned earlier. The Clock Event Page displays clock event activity, showing all 256 possible clock events, updated each cycle. The Download Page allows copying of LAs or PAs from one system to another, preserving the version date. It can also produce a directory listing of all "files" in a node, or of selected files via a name pattern filter, and send it to a given node's serial port. The Page G application allows viewing and control of another node's page display. The Ping Page provides a IP Ping client, UDP Echo client, Network Time Protocol client, and Domain Name Server client, all useful tools for an Internet Protocol environment.

The Setting Log Page shows time-stamped recent setting activity inside a given node by reading out that node's SETTING data stream queue. The Network Frames Page shows time-stamped recent network frame activity, either received or transmitted, by reading out a target node's NETFRAME data stream queue. This PA, sometimes referred to as a "poor man's Sniffer," has proven of immense value in tracking down network-related problems. A key feature is that the time-stamps are specified down to milliseconds into the cycle, important timing units for a synchronous control system. The Swift Commands Page shows the time-stamped recent Swift Digitizer commands by reading out the SWFTCMND data stream queue of a target node that has Swift Digitizer hardware.

The Memory Block Page shows allocated blocks of dynamic memory. This is one of the few PAs that must execute on the system in question. Simply use Page G to access that system, then invoke its Memory Block Page.

Test pages allow exercising the data request protocols supported by the system. The Request Timing Page exercises one-shot Classic protocol data requests, building a histogram of the response times. Here, local data access is obviously much faster than accessing data from another node. The RETDAT/SETDAT Page exercises those two Acnet protocols. The D0 Requests Page and D0 Settings Page exercise the protocol defined for use with the Fermilab D0 detector control system.

Network support

Network support for an IRM uses Internet Protocols on ethernet. (After all, IRM *means* Internet Rack Monitor.) Within the system support is also network support for non-IP communications both on token ring and also arcnet. But that support is normally not needed in an IRM. The rest of this section discusses only IP on ethernet.

Network support—IP layer

Internet protocols supported by an IRM system include ARP, IP, ICMP, UDP, and IGMP. Via local applications, support is provided for TFTP, NTP, and DNS protocols. High-level network communications software in an IRM refers to a node via a 16-bit node#. Since a UDP socket requires specification of a 32-bit IP address and a 16-bit port#, a range of pseudo node#s from `E000–EFFF` is

used to indirectly specify this information that is mapped via ARP table entries that include an IP address, a physical address, and a pointer to a block allocated to hold up to 15 active UDP port#s. In this way, the pseudo node# carries the meaning of an UDP socket.

Each IRM is assigned a node# in the range `05xx-07xx`. (This refers to the node itself, not a UDP socket.) To obtain the IP address for a given node, say node `0562`, the Fermilab Domain Name Server is accessed using the name `node0562.fnal.gov`. (For use elsewhere, an IRM can be configured to use a suffix other than `fnal.gov`.) The local application `DNSQ` provides this service; it caches the result in a system table, refreshing each cached entry every 8 hours just to prevent the cached entries from becoming stale. The very first time a given node is targeted, since the time the IRM's non-volatile memory was initially configured, the intended transmission will fail, since there was no available IP address in the cache. But after the prompt response from the DNS has been cached in the table, that target node will always be accessible from that IRM, because it will always have its IP address in the cache. This non-volatile table is disturbed neither by system resets nor by power cycles. (The table provides some of the functions of a "hosts" file on a workstation.) An entry can also be installed statically so that the DNS is not consulted to refresh it; this is especially useful for access to IRMs that are off-site.

Knowing an IP address of a node to be targeted is not enough to communicate with it; its 6-byte physical network address must be obtained as well. This is done using ARP (Address Resolution Protocol). If an ARP table entry is not currently active, an ARP request is sent to learn the physical address. Upon receipt of the ARP reply, queued messages are formed into frames for delivery. While awaiting an ARP reply from a given target node, network communications with other nodes are not impeded.

Included with IP is support for the ICMP (Internet Control Message Protocol). This supports the "ping" request commonly used to query a node's presence on the network. Another important ICMP message is one that means "port unreachable." If this message is received, the system automatically cancels replies for all data requests that originate from the node/port (socket) referred to as unreachable.

The Internet Group Message Protocol, or IGMP, is used to support IP multicasting. A multicast router, for each network to which it connects, sends an IGMP request every minute to the "all hosts" IP multicast address that is supported by all nodes that participate in IP multicasting. Each node that sees this request schedules, randomly over the next 10 seconds, an IGMP reply for each IP multicast address in which it has an interest. This reply is targeted to the same IP multicast address. The router is able to receive all such multicast replies. If a node receives such a multicast reply from another node, however, it will cancel its intention to send that same reply. In this way, only minimal network activity is required to keep a router informed of IP multicast interest on the part of all nodes on a network. (The router doesn't care how many nodes have an interest in a given IP multicast address, only that at least one has such interest.)

The User Datagram Protocol, or UDP, is the support that carries data request and alarm messages. It provides the same service as IP, but with the addition of a port#, so that a UDP port in one node can send a message to UDP port in another node. For some protocols, port#s have been standardized, including ECHO (UDP Echo Protocol), TFTP (Trivial File Transfer Protocol), NTP (Network Time Protocol), and DNS (Domain Name Service) protocol. Local or page applications can be designed as UDP protocol clients or servers. The IRM systems use fixed UDP port#s for the data request and alarm protocols they support. All Classic protocol messages target UDP port# 6800. All Acnet protocols target port# 6801. Acnet protocol support includes an additional layer that allows server support of a specific Acnet protocol via a local application.

Network support—messages

Network support software for applications is organized around messages, not frames. Messages are queued to a network, which means a pointer to an allocated message block is placed in a queue associated with that network. When the network queue is flushed, all queued messages are formatted into frames as required, without altering the order of messages in the queue. If consecutive messages that target the same node# are of the same type, say Classic protocol, they can be combined into common frames—in this case, UDP datagrams—as long as they will fit in the 9000-byte maximum size datagram that is supported, chosen so that one can include 8192 bytes of data in one message and then some. Of course, a datagram so large requires fragmentation, given the 1500-byte frame size limit of ethernet, so that support for fragmentation of outgoing datagrams and reassembly of incoming datagrams is included in IP support. If a program uses the network to send a datagram explicitly, say to issue a Ping request, the datagram is taken as a single message that cannot be combined with another. Message coalescing occurs when consecutive Classic or Acnet messages in the network queue target the same node. Maintaining the linked list of active request blocks ordered by target node# increases the likelihood of this happening, so that it is quite a common occurrence when multiple requests are active in a node—especially when a server node is used to forward the request messages to target node(s). (Note that the node#s targeted via IP are really pseudo node#s that specify an IP socket. This means that multiple UDP ports from a host node do not use the same target node#.) The message coalescing scheme allows the IRM to support a large number of active requests simultaneously and makes it easier on the host end, because the host doesn't have to receive so many network frames. Applications in the IRM acting as clients do not have to worry about this logic; it is supported by the underlying system automatically.

Data request protocols of two basic types are supported, both based upon UDP. (The D0 data request and alarm protocols are really just two of many in the Acnet protocol family.) In each case, the message types include data requests, data replies, setting requests, and alarm messages. Any of the message types may occur within the same datagram, as described above. The Classic protocol is the original network protocol supported by the system and is required to achieve full functionality with page applications. In fact, support for the Classic protocol is a major reason for the IRM's portability to non-Acnet control systems. The Acnet protocols are supported because they are required to serve the needs of the Fermilab accelerator control system. For both data request protocols, server support is provided. That means a node may receive a request for data that includes not only local device data, but device data that resides in other nodes. Server support arranges to forward the request to other nodes, using multicast addressing when appropriate, and upon receiving replies, build a composite reply for delivery to the original requester. The requesting node simply receives a reply to the data that was requested, without caring what node(s) actually sourced the data. For a project consisting of a dozen nodes, say, this can greatly reduce the number of datagrams a host must receive, especially when data requests ask for replies to be delivered at 10–15 Hz. Any IRM can provide server support for any request; they all run the same system software, so they all know how to do it.

Diagnostics

One diagnostic tool is the Macintosh Parameter Page. A Classic protocol client, it provides basic parameter page support. It also supports plots of device data and memory data, 1 KHz A/D data, an alarm display, and a Page G client.

A generally useful diagnostic has been the Memory Dump Page application discussed earlier. While it provides a view inside the memory of a running system, so that software problems can be diagnosed, it also has been useful for hardware diagnostics during development of a new board. Displaying memory data updated every cycle, accessed either by bytes or words, it also supports copying blocks of memory from any node to any node. Many particular areas in memory can be monitored, includ-

ing task-related variables for each task as well as the contents of various system tables, both static and dynamic. One has an inside view of what's going on in the system.

In a sense, all the page applications provide diagnostic or configuration support. If a problem arises in an analog channel, the Parameter Page application allows another view of accessing that channel independent of any host software. The Analog Page application shows all the fields in the local database for the channel and allows them to be changed as needed. The Clock Event Page shows what clock events are occurring. The Download Page helps in assembling the programs stored in the non-volatile memory file system. The Local Application Page is used to configure the arguments for all the local applications. It also shows the pointer to the executing LA's context memory, so that one can view such memory on the Memory Dump Page to understand how things look from the LA's point-of-view; in fact, when writing an LA, it may be useful to include diagnostics with this in mind. The Network Frame Page allows the capture of network frame activity diagnostics. Frame contents can also be captured with this page to check that a data request protocol, say, is formed correctly. The Page G emulator gains access to a remote station to execute and view a page application on that node. Normal IP diagnostics for the ICMP ping, UDP Echo, Network Time, and Domain Name Server are available from another page. One can view the internal allocated (volatile) blocks of memory to check for blocks that were not freed. The Setting Log Page views the log of recent setting activities. Data request protocols can be exercised for both Classic and Acnet protocols, including evaluating response time.

Lower-level diagnostics are available for capturing evidence of how a station died as a result of an unexpected exception, such as a bus error. Connecting an IRM to a terminal emulator, one can use the 162bug monitor program to display memory and disassemble 68K instructions in the event of a major crash, in order to determine what caused the crash. Fortunately, this doesn't often happen.

Hardware diagnostics are available via the task LEDs and interrupt routine LEDs on the front panel of an IRM. An LED is *on* during the execution of each task. An LED is *on* during the execution of each interrupt routine. Using a logic state analyzer, one can get accurate timing and sequencing information for all these activities. Task activities and interrupt activities together cover about all there is that goes on inside an IRM. These traces allow one to see first hand how well the IRM front end operates in strict real-time.

Appendix—History

The IRM software has a long history. Its earliest version supported the first distributed accelerator control system at Fermilab. The current version runs on either a 68020- or 68040-based hardware configuration.

Year	*#nodes*	*Network*	*CPU*	*Field bus*	*Installation*
1982	17	Ethernet	68000	SDLC	Fermilab 200 Mev Linac
1984	~	Arcnet	68010	VME	Michigan State Superconducting Cyclotron Lab
1986	~	Token ring	68020	1553	Loma Linda University Medical Center
1988	40	Token ring	68020	1553	Fermilab D0 Detector
1992	20	Token ring	68020	Arcnet	Fermilab 400 Mev Linac upgrade
1994	1	Ethernet	68040	IRM	BNL/RHIC Intrumentation group
1994	4	Ethernet	68040	IRM	DESY Tesla Modulator
1994	2	Ethernet	68040	IRM	Fermilab superconducting RF coupler
1994	1	Token ring	68020	1553	Fermilab superconducting RF cavity at A0
1996	21	Ethernet	68040	IRM	Fermilab Booster HLRF
1996	10	Ethernet	68040	IRM	Shreveport, LA, PET accelerator
1996	2	Ethernet	68040	IRM	Fermilab Recycler ring for Main Injector
1996	1	Ethernet	68040	IRM	Madison, WI, electron recycling exp't
1997	9	Ethernet	68040	IRM	Fermilab Photo-injector
1997	20	Ethernet	68040	IRM	Fermilab Main Injector HLRF